## CLAIMS

What is claimed is:

1. A method comprising:

re-compiling a function when a field watch for a field is activated, the function including a byte code sequence having a field byte code that accesses or modifies the field, the recompiled function providing a native code and occupying a code space;

generating an instrumentation code corresponding to the field watch of the field; and

inserting the instrumentation code to the native code.

2. The method of claim 1 further comprising:

guarding execution of the instrumentation code if the field watch is not activated.

3. The method of claim 1 wherein generating the instrumentation code comprises:

executing a field watch sequence if the field watch is activated.

4. The method of claim 1 wherein executing the field watch sequence comprises:

saving live global state, the live global state corresponding to an active register;

executing an event hook function for an event corresponding to the field watch; and

restoring the live global state.

5. The method of claim 4 wherein saving the live global state comprises:

pushing the live global state onto a stack.

6. The method of claim 4 wherein executing the event hook function comprises:

passing an argument corresponding to the field; and

calling a run-time library function related to the event.

7. The method of claim 5 wherein restoring the live global state comprises:

retrieving the live global state from the stack.

8. The method of claim 1 wherein inserting the instrumentation code comprises:

inserting the instrumentation code in a stub at end of the code space.

9. The method of claim 2 wherein guarding execution of the instrumentation code comprises:

updating an offset of a jump instruction to the stub when the field watch is activated.

10. The method of claim 1 wherein guarding execution of the instrumentation code comprises:

replacing a no-op sequence with a jump instruction to the stub.

11. The method of claim 9 further comprising:

clearing the field watch by replacing the offset with a zero offset.

12. The method of claim 10 further comprising:

clearing the field watch by replacing the jump instruction with the no-op sequence.

13. The method of claim 1 wherein the function is a Java method.

14. The method of claim 1 wherein the field is a Java field in a Java virtual machine.

15. The method of claim 4 wherein the event hook function is compatible with a Java Virtual Machine Debug Interface (JVMDI).

16. A computer program product comprising:

a machine useable medium having computer program code embedded therein, the computer program product having:

computer readable program code to re-compile a function when a field watch for a field is activated, the function including a byte code sequence having a field byte code that accesses or modifies the field, the recompiled function providing a native code and occupying a code space,

computer readable program code to generate an instrumentation code corresponding to the field watch of the field, and

computer readable program code to insert the instrumentation code to the native code.

17. The computer program product of claim 16 further comprising:

computer readable program code to guard execution of the instrumentation code if the field watch is not activated.

18. The computer program product of claim 16 wherein the computer readable program code to generate the instrumentation code comprises:

computer readable program code to execute a field watch sequence if the field watch is activated.

19. The computer program product of claim 16 wherein the computer readable program code to execute a field watch sequence comprises:

computer readable program code to save live global state, the live global state corresponding to an active register;

computer readable program code to execute an event hook function for an event corresponding to the field watch; and

computer readable program code to restore the live global state.

20. The computer program product of claim 19 wherein the computer readable program code to save the live global state comprises:

computer readable program code to push the live global state onto a stack.

21. The computer program product of claim 19 wherein the computer readable program code to execute the event hook function comprises:

computer readable program code to pass an argument corresponding to the field; and

computer readable program code to call a run-time library function related to the event.

22. The computer program product of claim 20 wherein the computer readable program code to restore the live global state comprises:

computer readable program code to retrieve the live global state from the stack.

23. The computer program product of claim 16 wherein the computer readable program code to insert the instrumentation code comprises:

computer readable program code to insert the instrumentation code in a stub at end of the code space.

24. The computer program product of claim 16 wherein the computer readable program code to guard execution of the instrumentation code comprises:

computer readable program code to update an offset of a jump instruction to the stub when the field watch is activated.

25. The computer program product of claim 16 wherein the computer readable program code to guard execution of the instrumentation code comprises:

computer readable program code to replace a no-op sequence with a jump instruction to the stub.

26. The computer program product of claim 24 further comprising:

computer readable program code to clear the field watch by replacing the offset with a zero offset.

27. The computer program product of claim 25 further comprising:

computer readable program code to clear the field watch by replacing the jump instruction with the no-op sequence.

28. The computer program product of claim 16 wherein the function is a Java method.

29. The computer program product of claim 16 wherein the field is a Java field in a Java virtual machine.

30. The computer program product of claim 19 wherein the event hook function is compatible with a Java Virtual Machine Debug Interface (JVMDI).

31. A system comprising:

a processor;

a memory coupled to the processor to store instruction code, the instruction code, when executed by the processor, causing the processor to:

re-compile a function when a field watch for a field is activated, the function including a byte code sequence having a field byte code that accesses or modifies the field, the re-compiled function providing a native code and occupying a code space,

generate an instrumentation code corresponding to the field watch of the field, and

insert the instrumentation code to the native code.

32. The system of claim 31 the instruction code further causing the processor to:

guard execution of the instrumentation code if the field watch is not activated.

33. The system of claim 31 wherein the instruction code causing the processor to generate the instrumentation code causes the processor to:

execute a field watch sequence if the field watch is activated.

34. The system of claim 31 wherein the instruction code causing the processor to execute a field watch sequence causes the processor to:

save live global state, the live global state corresponding to an active register;

execute an event hook function for an event corresponding to the field watch; and

restore the live global state.

35. The system of claim 32 wherein the instruction code causing the processor to guard execution of the instrumentation code causes the processor to:

update an offset of a jump instruction to the stub when the field watch is activated.

36. The system of claim 32 wherein the instruction code causing the processor to guard execution of the instrumentation code causes the processor to:

replace a no-op sequence with a jump instruction to the stub.

37. The system of claim 31 wherein the function is a Java method.

38. The system of claim 31 wherein the field is a Java field in a Java virtual machine.

39. The system of claim 34 wherein the event hook function is compatible with a Java Virtual Machine Debug Interface (JVMDI).